# Explication cellule-par-cellule (code seulement) - detection\_intention

**Ce document explique, ligne par ligne, toutes les cellules de code du notebook "detection\_intention.ipynb".** Seules les cellules de code sont couvertes; les cellules markdown sont ignorees.  
  
Pour chaque ligne :  
 - Une explication simple de ce que fait la ligne  
 - La raison du choix lorsque elle est detectee

## Cellule de code 1

**Code:**

# Verification GPU et installation des dependances  
  
import torch  
  
  
  
print('='\*70)  
  
print('DETECTION INTENTION - Transformers + CamemBERT')  
  
print('='\*70)  
  
  
  
print(f'\nPyTorch version: {torch.\_\_version\_\_}')  
  
print(f'CUDA disponible: {torch.cuda.is\_available()}')  
  
  
  
if torch.cuda.is\_available():  
  
 print(f'GPU: {torch.cuda.get\_device\_name(0)}')  
  
 print(f'Memoire GPU: {torch.cuda.get\_device\_properties(0).total\_memory / 1e9:.2f} GB')  
  
else:  
  
 print('ATTENTION: Pas de GPU detecte!')  
  
 print('Allez dans Runtime > Change runtime type > GPU')  
  
  
  
# Installation des packages Transformers  
  
!pip install -q transformers datasets evaluate seqeval accelerate scikit-learn langdetect  
  
print('\nInstallation terminee!')

Explications ligne par ligne :

**001 |** # Verification GPU et installation des dependances

**Explication:** Comment: Verification GPU et installation des dependances

**Raison / Choix:** Comment describing code intent.

**003 |** import torch

**Explication:** Import the PyTorch library.

**Raison / Choix:** Used for tensor operations and to check GPU availability and for model training.

**007 |** print('='\*70)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**009 |** print('DETECTION INTENTION - Transformers + CamemBERT')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**011 |** print('='\*70)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**015 |** print(f'\nPyTorch version: {torch.\_\_version\_\_}')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**017 |** print(f'CUDA disponible: {torch.cuda.is\_available()}')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**021 |** if torch.cuda.is\_available():

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**023 |**  print(f'GPU: {torch.cuda.get\_device\_name(0)}')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**025 |**  print(f'Memoire GPU: {torch.cuda.get\_device\_properties(0).total\_memory / 1e9:.2f} GB')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**027 |** else:

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**029 |**  print('ATTENTION: Pas de GPU detecte!')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**031 |**  print('Allez dans Runtime > Change runtime type > GPU')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**035 |** # Installation des packages Transformers

**Explication:** Comment: Installation des packages Transformers

**Raison / Choix:** Comment describing code intent.

**037 |** !pip install -q transformers datasets evaluate seqeval accelerate scikit-learn langdetect

**Explication:** Install Python packages in the Colab environment.

**Raison / Choix:** Ensures required dependencies (transformers, datasets, etc.) are available in the notebook runtime.

**039 |** print('\nInstallation terminee!')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

## Cellule de code 2

**Code:**

from google.colab import drive  
  
import os  
  
  
  
# Monter Google Drive  
  
drive.mount('/content/drive')  
  
  
  
# Desactiver WandB  
  
os.environ['WANDB\_DISABLED'] = 'true'  
  
  
  
# Chemins  
  
workdir = '/content/drive/MyDrive/dataset'  
  
os.makedirs(workdir, exist\_ok=True)  
  
  
  
print('Working directory:', workdir)  
  
print('WandB: DESACTIVE')

Explications ligne par ligne :

**001 |** from google.colab import drive

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**003 |** import os

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**007 |** # Monter Google Drive

**Explication:** Comment: Monter Google Drive

**Raison / Choix:** Comment describing code intent.

**009 |** drive.mount('/content/drive')

**Explication:** Mount Google Drive.

**Raison / Choix:** Allows reading and writing datasets/models on Google Drive in Colab.

**013 |** # Desactiver WandB

**Explication:** Comment: Desactiver WandB

**Raison / Choix:** Comment describing code intent.

**015 |** os.environ['WANDB\_DISABLED'] = 'true'

**Explication:** Disable Weights & Biases logging.

**Raison / Choix:** Avoids external experiment tracking to keep runs lightweight and private.

**019 |** # Chemins

**Explication:** Comment: Chemins

**Raison / Choix:** Comment describing code intent.

**021 |** workdir = '/content/drive/MyDrive/dataset'

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**023 |** os.makedirs(workdir, exist\_ok=True)

**Explication:** Create / join filesystem paths.

**Raison / Choix:** Set up working directories consistently (works on Colab and local).

**027 |** print('Working directory:', workdir)

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**029 |** print('WandB: DESACTIVE')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

## Cellule de code 3

**Code:**

import pandas as pd  
  
  
  
train\_path = os.path.join(workdir, 'train\_set.csv')  
  
test\_path = os.path.join(workdir, 'test\_set.csv')  
  
  
  
# Verification existence  
  
for p in [train\_path, test\_path]:  
  
 if not os.path.exists(p):  
  
 print(f'ERREUR: Fichier non trouve: {p}')  
  
 raise FileNotFoundError(p)  
  
  
  
# Chargement  
  
train\_df = pd.read\_csv(train\_path, encoding='utf-8')  
  
test\_df = pd.read\_csv(test\_path, encoding='utf-8')  
  
  
  
print(f'Train shape: {train\_df.shape}')  
  
print(f'Test shape: {test\_df.shape}')  
  
  
  
print('\nDistribution des classes (Train):')  
  
print(train\_df['intent'].value\_counts())  
  
print(f'\nPourcentages:')  
  
print(train\_df['intent'].value\_counts(normalize=True).apply(lambda x: f'{x:.2%}'))  
  
  
  
display(train\_df.head(3))

Explications ligne par ligne :

**001 |** import pandas as pd

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**005 |** train\_path = os.path.join(workdir, 'train\_set.csv')

**Explication:** Create / join filesystem paths.

**Raison / Choix:** Set up working directories consistently (works on Colab and local).

**007 |** test\_path = os.path.join(workdir, 'test\_set.csv')

**Explication:** Create / join filesystem paths.

**Raison / Choix:** Set up working directories consistently (works on Colab and local).

**011 |** # Verification existence

**Explication:** Comment: Verification existence

**Raison / Choix:** Comment describing code intent.

**013 |** for p in [train\_path, test\_path]:

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**015 |**  if not os.path.exists(p):

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**017 |**  print(f'ERREUR: Fichier non trouve: {p}')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**019 |**  raise FileNotFoundError(p)

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**023 |** # Chargement

**Explication:** Comment: Chargement

**Raison / Choix:** Comment describing code intent.

**025 |** train\_df = pd.read\_csv(train\_path, encoding='utf-8')

**Explication:** Load a CSV file into a pandas DataFrame.

**Raison / Choix:** Used to read train/test datasets; pandas is convenient for analysis and preprocessing.

**027 |** test\_df = pd.read\_csv(test\_path, encoding='utf-8')

**Explication:** Load a CSV file into a pandas DataFrame.

**Raison / Choix:** Used to read train/test datasets; pandas is convenient for analysis and preprocessing.

**031 |** print(f'Train shape: {train\_df.shape}')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**033 |** print(f'Test shape: {test\_df.shape}')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**037 |** print('\nDistribution des classes (Train):')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**039 |** print(train\_df['intent'].value\_counts())

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**041 |** print(f'\nPourcentages:')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**043 |** print(train\_df['intent'].value\_counts(normalize=True).apply(lambda x: f'{x:.2%}'))

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**047 |** display(train\_df.head(3))

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

## Cellule de code 4

**Code:**

import json  
  
  
  
# Nettoyer les colonnes necessaires  
  
train\_df = train\_df[['text', 'intent', 'entities']].dropna(subset=['text', 'intent']).reset\_index(drop=True)  
  
test\_df = test\_df[['text', 'intent', 'entities']].dropna(subset=['text', 'intent']).reset\_index(drop=True)  
  
  
  
def parse\_entities\_field(row):  
  
 """Parse la colonne entities (JSON) et valide."""  
  
 try:  
  
 ents = json.loads(row['entities']) if pd.notna(row['entities']) else []  
  
 except:  
  
 ents = []  
  
  
  
 valid = []  
  
 txt = row.get('text', '')  
  
 for ent in ents:  
  
 if isinstance(ent, dict) and 'start' in ent and 'end' in ent and 'label' in ent:  
  
 if 0 <= ent['start'] < ent['end'] <= len(txt):  
  
 valid.append(ent)  
  
 return valid  
  
  
  
# Parser les entities  
  
train\_df['parsed\_entities'] = train\_df.apply(parse\_entities\_field, axis=1)  
  
test\_df['parsed\_entities'] = test\_df.apply(parse\_entities\_field, axis=1)  
  
  
  
print('Entities parsees avec succes!')  
  
print(f'\nExemple TRIP avec entities:')  
  
trip\_ex = train\_df[train\_df['intent'] == 'TRIP'].iloc[0]  
  
print(f' Texte: {trip\_ex["text"]}')  
  
print(f' Entities: {trip\_ex["parsed\_entities"]}')

Explications ligne par ligne :

**001 |** import json

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**005 |** # Nettoyer les colonnes necessaires

**Explication:** Comment: Nettoyer les colonnes necessaires

**Raison / Choix:** Comment describing code intent.

**007 |** train\_df = train\_df[['text', 'intent', 'entities']].dropna(subset=['text', 'intent']).reset\_index(drop=True)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**009 |** test\_df = test\_df[['text', 'intent', 'entities']].dropna(subset=['text', 'intent']).reset\_index(drop=True)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**013 |** def parse\_entities\_field(row):

**Explication:** Parse JSON-encoded entity annotations.

**Raison / Choix:** Validates entity offsets for NER training and avoids corrupted annotations.

**015 |**  """Parse la colonne entities (JSON) et valide."""

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**017 |**  try:

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**019 |**  ents = json.loads(row['entities']) if pd.notna(row['entities']) else []

**Explication:** Parse JSON-encoded entity annotations.

**Raison / Choix:** Validates entity offsets for NER training and avoids corrupted annotations.

**021 |**  except:

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**023 |**  ents = []

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**027 |**  valid = []

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**029 |**  txt = row.get('text', '')

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**031 |**  for ent in ents:

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**033 |**  if isinstance(ent, dict) and 'start' in ent and 'end' in ent and 'label' in ent:

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**035 |**  if 0 <= ent['start'] < ent['end'] <= len(txt):

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**037 |**  valid.append(ent)

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**039 |**  return valid

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**043 |** # Parser les entities

**Explication:** Comment: Parser les entities

**Raison / Choix:** Comment describing code intent.

**045 |** train\_df['parsed\_entities'] = train\_df.apply(parse\_entities\_field, axis=1)

**Explication:** Parse JSON-encoded entity annotations.

**Raison / Choix:** Validates entity offsets for NER training and avoids corrupted annotations.

**047 |** test\_df['parsed\_entities'] = test\_df.apply(parse\_entities\_field, axis=1)

**Explication:** Parse JSON-encoded entity annotations.

**Raison / Choix:** Validates entity offsets for NER training and avoids corrupted annotations.

**051 |** print('Entities parsees avec succes!')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**053 |** print(f'\nExemple TRIP avec entities:')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**055 |** trip\_ex = train\_df[train\_df['intent'] == 'TRIP'].iloc[0]

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**057 |** print(f' Texte: {trip\_ex["text"]}')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**059 |** print(f' Entities: {trip\_ex["parsed\_entities"]}')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

## Cellule de code 5

**Code:**

from datasets import Dataset  
  
from sklearn.preprocessing import LabelEncoder  
  
  
  
# Encoder les labels d'intent  
  
label\_encoder = LabelEncoder()  
  
all\_intents = list(set(list(train\_df['intent'].unique()) + list(test\_df['intent'].unique())))  
  
label\_encoder.fit(all\_intents)  
  
  
  
print('Classes d\'intent:')  
  
for i, label in enumerate(label\_encoder.classes\_):  
  
 print(f' {i}: {label}')  
  
  
  
# Creer les datasets HuggingFace  
  
train\_df['label'] = label\_encoder.transform(train\_df['intent'])  
  
test\_df['label'] = label\_encoder.transform(test\_df['intent'])  
  
  
  
intent\_train\_dataset = Dataset.from\_pandas(train\_df[['text', 'label']])  
  
intent\_test\_dataset = Dataset.from\_pandas(test\_df[['text', 'label']])  
  
  
  
print(f'\nIntent datasets crees:')  
  
print(f' Train: {len(intent\_train\_dataset)} exemples')  
  
print(f' Test: {len(intent\_test\_dataset)} exemples')

Explications ligne par ligne :

**001 |** from datasets import Dataset

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**003 |** from sklearn.preprocessing import LabelEncoder

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**007 |** # Encoder les labels d'intent

**Explication:** Comment: Encoder les labels d'intent

**Raison / Choix:** Comment describing code intent.

**009 |** label\_encoder = LabelEncoder()

**Explication:** Encode string labels to integer IDs.

**Raison / Choix:** Required by ML frameworks which expect numeric labels for classification.

**011 |** all\_intents = list(set(list(train\_df['intent'].unique()) + list(test\_df['intent'].unique())))

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**013 |** label\_encoder.fit(all\_intents)

**Explication:** Encode string labels to integer IDs.

**Raison / Choix:** Required by ML frameworks which expect numeric labels for classification.

**017 |** print('Classes d\'intent:')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**019 |** for i, label in enumerate(label\_encoder.classes\_):

**Explication:** Encode string labels to integer IDs.

**Raison / Choix:** Required by ML frameworks which expect numeric labels for classification.

**021 |**  print(f' {i}: {label}')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**025 |** # Creer les datasets HuggingFace

**Explication:** Comment: Creer les datasets HuggingFace

**Raison / Choix:** Comment describing code intent.

**027 |** train\_df['label'] = label\_encoder.transform(train\_df['intent'])

**Explication:** Encode string labels to integer IDs.

**Raison / Choix:** Required by ML frameworks which expect numeric labels for classification.

**029 |** test\_df['label'] = label\_encoder.transform(test\_df['intent'])

**Explication:** Encode string labels to integer IDs.

**Raison / Choix:** Required by ML frameworks which expect numeric labels for classification.

**033 |** intent\_train\_dataset = Dataset.from\_pandas(train\_df[['text', 'label']])

**Explication:** Create a HuggingFace Dataset from a pandas DataFrame.

**Raison / Choix:** Datasets work well with the Trainer API and can be tokenized with map().

**035 |** intent\_test\_dataset = Dataset.from\_pandas(test\_df[['text', 'label']])

**Explication:** Create a HuggingFace Dataset from a pandas DataFrame.

**Raison / Choix:** Datasets work well with the Trainer API and can be tokenized with map().

**039 |** print(f'\nIntent datasets crees:')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**041 |** print(f' Train: {len(intent\_train\_dataset)} exemples')

**Explication:** Create a HuggingFace Dataset from a pandas DataFrame.

**Raison / Choix:** Datasets work well with the Trainer API and can be tokenized with map().

**043 |** print(f' Test: {len(intent\_test\_dataset)} exemples')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

## Cellule de code 6

**Code:**

from transformers import (  
  
 AutoTokenizer,  
  
 AutoModelForSequenceClassification,  
  
 TrainingArguments,  
  
 Trainer  
  
)  
  
from sklearn.utils.class\_weight import compute\_class\_weight  
  
import torch.nn as nn  
  
import evaluate  
  
import numpy as np  
  
  
  
# Tokenizer  
  
tokenizer = AutoTokenizer.from\_pretrained('camembert-base')  
  
  
  
def tokenize\_function(examples):  
  
 return tokenizer(  
  
 examples['text'],  
  
 padding='max\_length',  
  
 truncation=True,  
  
 max\_length=128  
  
 )  
  
  
  
print('Tokenization...')  
  
tokenized\_train = intent\_train\_dataset.map(tokenize\_function, batched=True)  
  
tokenized\_test = intent\_test\_dataset.map(tokenize\_function, batched=True)  
  
print('Tokenization terminee!')  
  
  
  
# Modele avec dropout augmente  
  
num\_labels = len(label\_encoder.classes\_)  
  
intent\_model = AutoModelForSequenceClassification.from\_pretrained(  
  
 'camembert-base',  
  
 num\_labels=num\_labels,  
  
 id2label={i: label for i, label in enumerate(label\_encoder.classes\_)},  
  
 label2id={label: i for i, label in enumerate(label\_encoder.classes\_)},  
  
 hidden\_dropout\_prob=0.4,  
  
 attention\_probs\_dropout\_prob=0.4,  
  
 classifier\_dropout=0.4  
  
)  
  
  
  
# Class weights  
  
class\_weights = compute\_class\_weight(  
  
 class\_weight='balanced',  
  
 classes=np.unique(train\_df['label']),  
  
 y=train\_df['label']  
  
)  
  
  
  
print('\nClass weights:')  
  
for label, weight in zip(label\_encoder.classes\_, class\_weights):  
  
 print(f' {label}: {weight:.3f}')  
  
  
  
# Trainer avec weighted loss (CORRIGE pour nouvelle API Transformers)  
  
class WeightedTrainer(Trainer):  
  
 def compute\_loss(self, model, inputs, return\_outputs=False, num\_items\_in\_batch=None):  
  
 labels = inputs.get("labels")  
  
 outputs = model(\*\*inputs)  
  
 logits = outputs.get("logits")  
  
  
  
 weight\_tensor = torch.tensor(class\_weights, dtype=torch.float).to(logits.device)  
  
 loss\_fct = nn.CrossEntropyLoss(weight=weight\_tensor)  
  
 loss = loss\_fct(logits.view(-1, self.model.config.num\_labels), labels.view(-1))  
  
  
  
 return (loss, outputs) if return\_outputs else loss  
  
  
  
# Metriques  
  
accuracy\_metric = evaluate.load('accuracy')  
  
f1\_metric = evaluate.load('f1')  
  
  
  
def compute\_intent\_metrics(eval\_pred):  
  
 logits, labels = eval\_pred  
  
 predictions = np.argmax(logits, axis=-1)  
  
  
  
 from sklearn.metrics import f1\_score, precision\_score, recall\_score  
  
  
  
 accuracy = (predictions == labels).mean()  
  
 f1\_macro = f1\_score(labels, predictions, average='macro')  
  
 f1\_per\_class = f1\_score(labels, predictions, average=None)  
  
  
  
 metrics = {  
  
 'accuracy': float(accuracy),  
  
 'f1\_macro': float(f1\_macro)  
  
 }  
  
  
  
 for i, label in enumerate(label\_encoder.classes\_):  
  
 metrics[f'f1\_{label}'] = float(f1\_per\_class[i])  
  
  
  
 return metrics  
  
  
  
# Training arguments optimises  
  
training\_args = TrainingArguments(  
  
 output\_dir=os.path.join(workdir, 'models/intent\_classifier'),  
  
 num\_train\_epochs=6,  
  
 per\_device\_train\_batch\_size=16,  
  
 per\_device\_eval\_batch\_size=32,  
  
 learning\_rate=8e-6,  
  
 weight\_decay=0.03,  
  
 eval\_strategy='epoch',  
  
 save\_strategy='epoch',  
  
 load\_best\_model\_at\_end=True,  
  
 metric\_for\_best\_model='f1\_macro',  
  
 logging\_steps=50,  
  
 warmup\_steps=300,  
  
 fp16=torch.cuda.is\_available(),  
  
 push\_to\_hub=False,  
  
 report\_to='none',  
  
 gradient\_accumulation\_steps=2,  
  
)  
  
  
  
# Trainer  
  
intent\_trainer = WeightedTrainer(  
  
 model=intent\_model,  
  
 args=training\_args,  
  
 train\_dataset=tokenized\_train,  
  
 eval\_dataset=tokenized\_test,  
  
 tokenizer=tokenizer,  
  
 compute\_metrics=compute\_intent\_metrics  
  
)  
  
  
  
print('\n' + '='\*70)  
  
print('DEBUT DU FINE-TUNING AMELIORE')  
  
print('='\*70)  
  
  
  
intent\_trainer.train()  
  
  
  
print('\nFine-tuning termine!')

Explications ligne par ligne :

**001 |** from transformers import (

**Explication:** Import Transformers components.

**Raison / Choix:** We use Hugging Face Transformers for tokenization, modelling and training.

**003 |**  AutoTokenizer,

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**005 |**  AutoModelForSequenceClassification,

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**007 |**  TrainingArguments,

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**009 |**  Trainer

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**011 |** )

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**013 |** from sklearn.utils.class\_weight import compute\_class\_weight

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**015 |** import torch.nn as nn

**Explication:** Import the PyTorch library.

**Raison / Choix:** Used for tensor operations and to check GPU availability and for model training.

**017 |** import evaluate

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**019 |** import numpy as np

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**023 |** # Tokenizer

**Explication:** Comment: Tokenizer

**Raison / Choix:** Comment describing code intent.

**025 |** tokenizer = AutoTokenizer.from\_pretrained('camembert-base')

**Explication:** Load a pretrained tokenizer (CamemBERT).

**Raison / Choix:** Tokenization converts raw text to token ids compatible with the model; CamemBERT is a French pretrained model.

**029 |** def tokenize\_function(examples):

**Explication:** Define a tokenization function for the dataset.

**Raison / Choix:** Centralizes tokenization parameters like max length and truncation for consistent preprocessing.

**031 |**  return tokenizer(

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**033 |**  examples['text'],

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**035 |**  padding='max\_length',

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**037 |**  truncation=True,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**039 |**  max\_length=128

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**041 |**  )

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**045 |** print('Tokenization...')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**047 |** tokenized\_train = intent\_train\_dataset.map(tokenize\_function, batched=True)

**Explication:** Create a HuggingFace Dataset from a pandas DataFrame.

**Raison / Choix:** Datasets work well with the Trainer API and can be tokenized with map().

**049 |** tokenized\_test = intent\_test\_dataset.map(tokenize\_function, batched=True)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**051 |** print('Tokenization terminee!')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**055 |** # Modele avec dropout augmente

**Explication:** Comment: Modele avec dropout augmente

**Raison / Choix:** Comment describing code intent.

**057 |** num\_labels = len(label\_encoder.classes\_)

**Explication:** Encode string labels to integer IDs.

**Raison / Choix:** Required by ML frameworks which expect numeric labels for classification.

**059 |** intent\_model = AutoModelForSequenceClassification.from\_pretrained(

**Explication:** Load a pretrained sequence classification model (CamemBERT).

**Raison / Choix:** We fine-tune a pretrained model to save training time and improve performance on French text.

**061 |**  'camembert-base',

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**063 |**  num\_labels=num\_labels,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**065 |**  id2label={i: label for i, label in enumerate(label\_encoder.classes\_)},

**Explication:** Encode string labels to integer IDs.

**Raison / Choix:** Required by ML frameworks which expect numeric labels for classification.

**067 |**  label2id={label: i for i, label in enumerate(label\_encoder.classes\_)},

**Explication:** Encode string labels to integer IDs.

**Raison / Choix:** Required by ML frameworks which expect numeric labels for classification.

**069 |**  hidden\_dropout\_prob=0.4,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**071 |**  attention\_probs\_dropout\_prob=0.4,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**073 |**  classifier\_dropout=0.4

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**075 |** )

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**079 |** # Class weights

**Explication:** Comment: Class weights

**Raison / Choix:** Comment describing code intent.

**081 |** class\_weights = compute\_class\_weight(

**Explication:** Compute class weights for imbalanced classes.

**Raison / Choix:** Gives more importance to minority classes during loss computation to reduce bias.

**083 |**  class\_weight='balanced',

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**085 |**  classes=np.unique(train\_df['label']),

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**087 |**  y=train\_df['label']

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**089 |** )

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**093 |** print('\nClass weights:')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**095 |** for label, weight in zip(label\_encoder.classes\_, class\_weights):

**Explication:** Encode string labels to integer IDs.

**Raison / Choix:** Required by ML frameworks which expect numeric labels for classification.

**097 |**  print(f' {label}: {weight:.3f}')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**101 |** # Trainer avec weighted loss (CORRIGE pour nouvelle API Transformers)

**Explication:** Comment: Trainer avec weighted loss (CORRIGE pour nouvelle API Transformers)

**Raison / Choix:** Comment describing code intent.

**103 |** class WeightedTrainer(Trainer):

**Explication:** Custom Trainer using a weighted loss.

**Raison / Choix:** Applies class weights to the loss so the model treats minority classes fairly.

**105 |**  def compute\_loss(self, model, inputs, return\_outputs=False, num\_items\_in\_batch=None):

**Explication:** Define function.

**Raison / Choix:** Creates a reusable function used later in the notebook.

**107 |**  labels = inputs.get("labels")

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**109 |**  outputs = model(\*\*inputs)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**111 |**  logits = outputs.get("logits")

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**115 |**  weight\_tensor = torch.tensor(class\_weights, dtype=torch.float).to(logits.device)

**Explication:** Compute class weights for imbalanced classes.

**Raison / Choix:** Gives more importance to minority classes during loss computation to reduce bias.

**117 |**  loss\_fct = nn.CrossEntropyLoss(weight=weight\_tensor)

**Explication:** Custom Trainer using a weighted loss.

**Raison / Choix:** Applies class weights to the loss so the model treats minority classes fairly.

**119 |**  loss = loss\_fct(logits.view(-1, self.model.config.num\_labels), labels.view(-1))

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**123 |**  return (loss, outputs) if return\_outputs else loss

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**127 |** # Metriques

**Explication:** Comment: Metriques

**Raison / Choix:** Comment describing code intent.

**129 |** accuracy\_metric = evaluate.load('accuracy')

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**131 |** f1\_metric = evaluate.load('f1')

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**135 |** def compute\_intent\_metrics(eval\_pred):

**Explication:** Define function.

**Raison / Choix:** Creates a reusable function used later in the notebook.

**137 |**  logits, labels = eval\_pred

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**139 |**  predictions = np.argmax(logits, axis=-1)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**143 |**  from sklearn.metrics import f1\_score, precision\_score, recall\_score

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**147 |**  accuracy = (predictions == labels).mean()

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**149 |**  f1\_macro = f1\_score(labels, predictions, average='macro')

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**151 |**  f1\_per\_class = f1\_score(labels, predictions, average=None)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**155 |**  metrics = {

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**157 |**  'accuracy': float(accuracy),

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**159 |**  'f1\_macro': float(f1\_macro)

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**161 |**  }

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**165 |**  for i, label in enumerate(label\_encoder.classes\_):

**Explication:** Encode string labels to integer IDs.

**Raison / Choix:** Required by ML frameworks which expect numeric labels for classification.

**167 |**  metrics[f'f1\_{label}'] = float(f1\_per\_class[i])

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**171 |**  return metrics

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**175 |** # Training arguments optimises

**Explication:** Comment: Training arguments optimises

**Raison / Choix:** Comment describing code intent.

**177 |** training\_args = TrainingArguments(

**Explication:** Define training hyperparameters.

**Raison / Choix:** Controls epochs, batch size, learning rate and other aspects of training stability and performance.

**179 |**  output\_dir=os.path.join(workdir, 'models/intent\_classifier'),

**Explication:** Create / join filesystem paths.

**Raison / Choix:** Set up working directories consistently (works on Colab and local).

**181 |**  num\_train\_epochs=6,

**Explication:** Define training hyperparameters.

**Raison / Choix:** Controls epochs, batch size, learning rate and other aspects of training stability and performance.

**183 |**  per\_device\_train\_batch\_size=16,

**Explication:** Define training hyperparameters.

**Raison / Choix:** Controls epochs, batch size, learning rate and other aspects of training stability and performance.

**185 |**  per\_device\_eval\_batch\_size=32,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**187 |**  learning\_rate=8e-6,

**Explication:** Define training hyperparameters.

**Raison / Choix:** Controls epochs, batch size, learning rate and other aspects of training stability and performance.

**189 |**  weight\_decay=0.03,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**191 |**  eval\_strategy='epoch',

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**193 |**  save\_strategy='epoch',

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**195 |**  load\_best\_model\_at\_end=True,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**197 |**  metric\_for\_best\_model='f1\_macro',

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**199 |**  logging\_steps=50,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**201 |**  warmup\_steps=300,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**203 |**  fp16=torch.cuda.is\_available(),

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**205 |**  push\_to\_hub=False,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**207 |**  report\_to='none',

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**209 |**  gradient\_accumulation\_steps=2,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**211 |** )

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**215 |** # Trainer

**Explication:** Comment: Trainer

**Raison / Choix:** Comment describing code intent.

**217 |** intent\_trainer = WeightedTrainer(

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**219 |**  model=intent\_model,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**221 |**  args=training\_args,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**223 |**  train\_dataset=tokenized\_train,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**225 |**  eval\_dataset=tokenized\_test,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**227 |**  tokenizer=tokenizer,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**229 |**  compute\_metrics=compute\_intent\_metrics

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**231 |** )

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**235 |** print('\n' + '='\*70)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**237 |** print('DEBUT DU FINE-TUNING AMELIORE')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**239 |** print('='\*70)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**243 |** intent\_trainer.train()

**Explication:** Start model fine-tuning.

**Raison / Choix:** Runs gradient updates on training data to adapt the pretrained model to the target task.

**247 |** print('\nFine-tuning termine!')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

## Cellule de code 7

**Code:**

from sklearn.metrics import classification\_report, confusion\_matrix  
  
import matplotlib.pyplot as plt  
  
import seaborn as sns  
  
  
  
# Evaluer sur le test set  
  
eval\_results = intent\_trainer.evaluate()  
  
  
  
print('='\*70)  
  
print('RESULTATS SUR LE TEST SET')  
  
print('='\*70)  
  
for key, value in eval\_results.items():  
  
 if isinstance(value, float):  
  
 print(f' {key}: {value:.4f}')  
  
  
  
# Predictions detaillees  
  
predictions = intent\_trainer.predict(tokenized\_test)  
  
predicted\_labels = np.argmax(predictions.predictions, axis=-1)  
  
true\_labels = predictions.label\_ids  
  
  
  
  
  
# Ajouter post-processing  
  
def post\_process\_prediction(text, predicted\_intent):  
  
 """  
  
 Corrige les predictions aberrantes avec regles heuristiques.  
  
 """  
  
 text\_lower = text.lower().strip()  
  
  
  
 # Detection langue  
  
 try:  
  
 from langdetect import detect  
  
 lang = detect(text) if len(text) >= 3 else 'unknown'  
  
 except:  
  
 lang = 'fr' # Par defaut francais  
  
  
  
 # REGLE 1: Langue non-francaise  
  
 english\_markers = ['i ', 'you ', 'the ', 'is ', 'are ', 'have ', 'do ', 'can ', 'hello', 'please']  
  
 spanish\_markers = ['el ', 'la ', 'los ', 'de ', 'que ', 'por ', 'para ', 'bueno', 'gracias']  
  
 german\_markers = ['der ', 'die ', 'das ', 'ist ', 'ich ', 'sie ', 'haben ', 'guten', 'danke']  
  
  
  
 if (lang not in ['fr', 'unknown']) or \  
  
 any(marker in text\_lower for marker in english\_markers + spanish\_markers + german\_markers):  
  
 if predicted\_intent != 'NOT\_FRENCH':  
  
 return 'NOT\_FRENCH'  
  
  
  
 # REGLE 2: Mots-cles NOT\_TRIP  
  
 not\_trip\_keywords = [  
  
 'merci', 'remercie', 'confirme', 'email', 'reunion', 'document',  
  
 'rapport', 'felicitation', 'bravo', 'compte-rendu', 'transferer'  
  
 ]  
  
 if any(kw in text\_lower for kw in not\_trip\_keywords):  
  
 trip\_keywords = ['billet', 'train', 'aller', 'retour', 'trajet']  
  
 has\_trip = any(kw in text\_lower for kw in trip\_keywords)  
  
 if not has\_trip and predicted\_intent == 'TRIP':  
  
 return 'NOT\_TRIP'  
  
  
  
 # REGLE 3: Texte incomprehensible (tres court, pas de mots francais)  
  
 if len(text\_lower) < 15 and lang == 'unknown':  
  
 words = text\_lower.split()  
  
 if len(words) <= 3:  
  
 # Verifier si ce sont des mots francais reconnaissables  
  
 french\_common = ['de', 'a', 'le', 'la', 'pour', 'je', 'tu', 'il']  
  
 if not any(word in french\_common for word in words):  
  
 if predicted\_intent != 'UNKNOWN':  
  
 return 'UNKNOWN'  
  
  
  
 # REGLE 4: Format "Ville1 Ville2" = TRIP  
  
 words = text\_lower.split()  
  
 if len(words) == 2 and len(text) < 50:  
  
 # Verifier si ce sont deux noms propres (majuscules)  
  
 if text.split()[0][0].isupper() and text.split()[1][0].isupper():  
  
 if predicted\_intent != 'TRIP':  
  
 return 'TRIP'  
  
  
  
 # REGLE 5: Presence "de X a Y" ou "X Y" = TRIP  
  
 if ('de ' in text\_lower and ' a ' in text\_lower) or \  
  
 ('de ' in text\_lower and ' vers ' in text\_lower) or \  
  
 (' pour ' in text\_lower and len(words) < 10):  
  
 trip\_keywords = ['billet', 'train', 'horaire', 'tarif']  
  
 not\_trip\_keywords\_strict = ['merci', 'document', 'rapport']  
  
 has\_not\_trip = any(kw in text\_lower for kw in not\_trip\_keywords\_strict)  
  
 if not has\_not\_trip and predicted\_intent != 'TRIP':  
  
 return 'TRIP'  
  
  
  
 return predicted\_intent  
  
  
  
# Appliquer post-processing  
  
test\_df\_eval = test\_df.copy()  
  
test\_df\_eval['predicted\_label'] = predicted\_labels  
  
test\_df\_eval['predicted\_intent'] = label\_encoder.inverse\_transform(predicted\_labels)  
  
  
  
corrected\_intents = []  
  
for idx, row in test\_df\_eval.iterrows():  
  
 corrected = post\_process\_prediction(row['text'], row['predicted\_intent'])  
  
 corrected\_intents.append(corrected)  
  
  
  
test\_df\_eval['corrected\_intent'] = corrected\_intents  
  
test\_df\_eval['corrected\_label'] = label\_encoder.transform(corrected\_intents)  
  
  
  
# Comparer  
  
accuracy\_before = (test\_df\_eval['label'] == test\_df\_eval['predicted\_label']).mean()  
  
accuracy\_after = (test\_df\_eval['label'] == test\_df\_eval['corrected\_label']).mean()  
  
  
  
print(f'\\nAccuracy avant post-processing: {accuracy\_before:.4f}')  
  
print(f'Accuracy apres post-processing: {accuracy\_after:.4f}')  
  
print(f'Amelioration: {(accuracy\_after - accuracy\_before):.4f} ({(accuracy\_after/accuracy\_before - 1)\*100:+.2f}%)')  
  
  
  
# Classification report apres PP  
  
from sklearn.metrics import classification\_report  
  
print('\\n' + '='\*70)  
  
print('CLASSIFICATION REPORT APRES POST-PROCESSING')  
  
print('='\*70)  
  
print(classification\_report(  
  
 test\_df\_eval['label'],  
  
 test\_df\_eval['corrected\_label'],  
  
 target\_names=label\_encoder.classes\_,  
  
 digits=4  
  
))  
  
  
  
# Classification report  
  
print('\n' + '='\*70)  
  
print('CLASSIFICATION REPORT')  
  
print('='\*70)  
  
print(classification\_report(  
  
 true\_labels,  
  
 predicted\_labels,  
  
 target\_names=label\_encoder.classes\_,  
  
 digits=4  
  
))  
  
  
  
# Matrice de confusion  
  
cm = confusion\_matrix(true\_labels, predicted\_labels)  
  
plt.figure(figsize=(10, 8))  
  
sns.heatmap(  
  
 cm,  
  
 annot=True,  
  
 fmt='d',  
  
 cmap='Blues',  
  
 xticklabels=label\_encoder.classes\_,  
  
 yticklabels=label\_encoder.classes\_  
  
)  
  
plt.title('Matrice de Confusion - Intent Classification')  
  
plt.ylabel('Vraie Classe')  
  
plt.xlabel('Classe Predite')  
  
plt.tight\_layout()  
  
plt.show()  
  
  
  
# Sauvegarder le modele  
  
model\_path = os.path.join(workdir, 'models/intent\_classifier\_best')  
  
intent\_trainer.save\_model(model\_path)  
  
tokenizer.save\_pretrained(model\_path)  
  
  
  
print(f'\nModele sauvegarde dans: {model\_path}')

Explications ligne par ligne :

**001 |** from sklearn.metrics import classification\_report, confusion\_matrix

**Explication:** Evaluate the model on the test set and compute metrics.

**Raison / Choix:** Gives objective performance measures like accuracy and F1 to assess model quality.

**003 |** import matplotlib.pyplot as plt

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**005 |** import seaborn as sns

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**009 |** # Evaluer sur le test set

**Explication:** Comment: Evaluer sur le test set

**Raison / Choix:** Comment describing code intent.

**011 |** eval\_results = intent\_trainer.evaluate()

**Explication:** Evaluate the model on the test set and compute metrics.

**Raison / Choix:** Gives objective performance measures like accuracy and F1 to assess model quality.

**015 |** print('='\*70)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**017 |** print('RESULTATS SUR LE TEST SET')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**019 |** print('='\*70)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**021 |** for key, value in eval\_results.items():

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**023 |**  if isinstance(value, float):

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**025 |**  print(f' {key}: {value:.4f}')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**029 |** # Predictions detaillees

**Explication:** Comment: Predictions detaillees

**Raison / Choix:** Comment describing code intent.

**031 |** predictions = intent\_trainer.predict(tokenized\_test)

**Explication:** Evaluate the model on the test set and compute metrics.

**Raison / Choix:** Gives objective performance measures like accuracy and F1 to assess model quality.

**033 |** predicted\_labels = np.argmax(predictions.predictions, axis=-1)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**035 |** true\_labels = predictions.label\_ids

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**041 |** # Ajouter post-processing

**Explication:** Comment: Ajouter post-processing

**Raison / Choix:** Comment describing code intent.

**043 |** def post\_process\_prediction(text, predicted\_intent):

**Explication:** Define post-processing heuristics for intent predictions.

**Raison / Choix:** Rules correct systematic errors (language detection, keywords) that the classifier alone may mishandle.

**045 |**  """

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**047 |**  Corrige les predictions aberrantes avec regles heuristiques.

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**049 |**  """

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**051 |**  text\_lower = text.lower().strip()

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**055 |**  # Detection langue

**Explication:** Comment: Detection langue

**Raison / Choix:** Comment describing code intent.

**057 |**  try:

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**059 |**  from langdetect import detect

**Explication:** Detect text language.

**Raison / Choix:** Used in post-processing to separate non-French queries (NOT\_FRENCH) from French ones.

**061 |**  lang = detect(text) if len(text) >= 3 else 'unknown'

**Explication:** Detect text language.

**Raison / Choix:** Used in post-processing to separate non-French queries (NOT\_FRENCH) from French ones.

**063 |**  except:

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**065 |**  lang = 'fr' # Par defaut francais

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**069 |**  # REGLE 1: Langue non-francaise

**Explication:** Comment: REGLE 1: Langue non-francaise

**Raison / Choix:** Comment describing code intent.

**071 |**  english\_markers = ['i ', 'you ', 'the ', 'is ', 'are ', 'have ', 'do ', 'can ', 'hello', 'please']

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**073 |**  spanish\_markers = ['el ', 'la ', 'los ', 'de ', 'que ', 'por ', 'para ', 'bueno', 'gracias']

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**075 |**  german\_markers = ['der ', 'die ', 'das ', 'ist ', 'ich ', 'sie ', 'haben ', 'guten', 'danke']

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**079 |**  if (lang not in ['fr', 'unknown']) or \

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**081 |**  any(marker in text\_lower for marker in english\_markers + spanish\_markers + german\_markers):

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**083 |**  if predicted\_intent != 'NOT\_FRENCH':

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**085 |**  return 'NOT\_FRENCH'

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**089 |**  # REGLE 2: Mots-cles NOT\_TRIP

**Explication:** Comment: REGLE 2: Mots-cles NOT\_TRIP

**Raison / Choix:** Comment describing code intent.

**091 |**  not\_trip\_keywords = [

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**093 |**  'merci', 'remercie', 'confirme', 'email', 'reunion', 'document',

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**095 |**  'rapport', 'felicitation', 'bravo', 'compte-rendu', 'transferer'

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**097 |**  ]

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**099 |**  if any(kw in text\_lower for kw in not\_trip\_keywords):

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**101 |**  trip\_keywords = ['billet', 'train', 'aller', 'retour', 'trajet']

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**103 |**  has\_trip = any(kw in text\_lower for kw in trip\_keywords)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**105 |**  if not has\_trip and predicted\_intent == 'TRIP':

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**107 |**  return 'NOT\_TRIP'

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**111 |**  # REGLE 3: Texte incomprehensible (tres court, pas de mots francais)

**Explication:** Comment: REGLE 3: Texte incomprehensible (tres court, pas de mots francais)

**Raison / Choix:** Comment describing code intent.

**113 |**  if len(text\_lower) < 15 and lang == 'unknown':

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**115 |**  words = text\_lower.split()

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**117 |**  if len(words) <= 3:

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**119 |**  # Verifier si ce sont des mots francais reconnaissables

**Explication:** Comment: Verifier si ce sont des mots francais reconnaissables

**Raison / Choix:** Comment describing code intent.

**121 |**  french\_common = ['de', 'a', 'le', 'la', 'pour', 'je', 'tu', 'il']

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**123 |**  if not any(word in french\_common for word in words):

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**125 |**  if predicted\_intent != 'UNKNOWN':

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**127 |**  return 'UNKNOWN'

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**131 |**  # REGLE 4: Format "Ville1 Ville2" = TRIP

**Explication:** Comment: REGLE 4: Format "Ville1 Ville2" = TRIP

**Raison / Choix:** Comment describing code intent.

**133 |**  words = text\_lower.split()

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**135 |**  if len(words) == 2 and len(text) < 50:

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**137 |**  # Verifier si ce sont deux noms propres (majuscules)

**Explication:** Comment: Verifier si ce sont deux noms propres (majuscules)

**Raison / Choix:** Comment describing code intent.

**139 |**  if text.split()[0][0].isupper() and text.split()[1][0].isupper():

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**141 |**  if predicted\_intent != 'TRIP':

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**143 |**  return 'TRIP'

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**147 |**  # REGLE 5: Presence "de X a Y" ou "X Y" = TRIP

**Explication:** Comment: REGLE 5: Presence "de X a Y" ou "X Y" = TRIP

**Raison / Choix:** Comment describing code intent.

**149 |**  if ('de ' in text\_lower and ' a ' in text\_lower) or \

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**151 |**  ('de ' in text\_lower and ' vers ' in text\_lower) or \

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**153 |**  (' pour ' in text\_lower and len(words) < 10):

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**155 |**  trip\_keywords = ['billet', 'train', 'horaire', 'tarif']

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**157 |**  not\_trip\_keywords\_strict = ['merci', 'document', 'rapport']

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**159 |**  has\_not\_trip = any(kw in text\_lower for kw in not\_trip\_keywords\_strict)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**161 |**  if not has\_not\_trip and predicted\_intent != 'TRIP':

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**163 |**  return 'TRIP'

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**167 |**  return predicted\_intent

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**171 |** # Appliquer post-processing

**Explication:** Comment: Appliquer post-processing

**Raison / Choix:** Comment describing code intent.

**173 |** test\_df\_eval = test\_df.copy()

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**175 |** test\_df\_eval['predicted\_label'] = predicted\_labels

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**177 |** test\_df\_eval['predicted\_intent'] = label\_encoder.inverse\_transform(predicted\_labels)

**Explication:** Encode string labels to integer IDs.

**Raison / Choix:** Required by ML frameworks which expect numeric labels for classification.

**181 |** corrected\_intents = []

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**183 |** for idx, row in test\_df\_eval.iterrows():

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**185 |**  corrected = post\_process\_prediction(row['text'], row['predicted\_intent'])

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**187 |**  corrected\_intents.append(corrected)

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**191 |** test\_df\_eval['corrected\_intent'] = corrected\_intents

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**193 |** test\_df\_eval['corrected\_label'] = label\_encoder.transform(corrected\_intents)

**Explication:** Encode string labels to integer IDs.

**Raison / Choix:** Required by ML frameworks which expect numeric labels for classification.

**197 |** # Comparer

**Explication:** Comment: Comparer

**Raison / Choix:** Comment describing code intent.

**199 |** accuracy\_before = (test\_df\_eval['label'] == test\_df\_eval['predicted\_label']).mean()

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**201 |** accuracy\_after = (test\_df\_eval['label'] == test\_df\_eval['corrected\_label']).mean()

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**205 |** print(f'\\nAccuracy avant post-processing: {accuracy\_before:.4f}')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**207 |** print(f'Accuracy apres post-processing: {accuracy\_after:.4f}')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**209 |** print(f'Amelioration: {(accuracy\_after - accuracy\_before):.4f} ({(accuracy\_after/accuracy\_before - 1)\*100:+.2f}%)')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**213 |** # Classification report apres PP

**Explication:** Comment: Classification report apres PP

**Raison / Choix:** Comment describing code intent.

**215 |** from sklearn.metrics import classification\_report

**Explication:** Evaluate the model on the test set and compute metrics.

**Raison / Choix:** Gives objective performance measures like accuracy and F1 to assess model quality.

**217 |** print('\\n' + '='\*70)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**219 |** print('CLASSIFICATION REPORT APRES POST-PROCESSING')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**221 |** print('='\*70)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**223 |** print(classification\_report(

**Explication:** Evaluate the model on the test set and compute metrics.

**Raison / Choix:** Gives objective performance measures like accuracy and F1 to assess model quality.

**225 |**  test\_df\_eval['label'],

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**227 |**  test\_df\_eval['corrected\_label'],

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**229 |**  target\_names=label\_encoder.classes\_,

**Explication:** Encode string labels to integer IDs.

**Raison / Choix:** Required by ML frameworks which expect numeric labels for classification.

**231 |**  digits=4

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**233 |** ))

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**237 |** # Classification report

**Explication:** Comment: Classification report

**Raison / Choix:** Comment describing code intent.

**239 |** print('\n' + '='\*70)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**241 |** print('CLASSIFICATION REPORT')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**243 |** print('='\*70)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**245 |** print(classification\_report(

**Explication:** Evaluate the model on the test set and compute metrics.

**Raison / Choix:** Gives objective performance measures like accuracy and F1 to assess model quality.

**247 |**  true\_labels,

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**249 |**  predicted\_labels,

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**251 |**  target\_names=label\_encoder.classes\_,

**Explication:** Encode string labels to integer IDs.

**Raison / Choix:** Required by ML frameworks which expect numeric labels for classification.

**253 |**  digits=4

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**255 |** ))

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**259 |** # Matrice de confusion

**Explication:** Comment: Matrice de confusion

**Raison / Choix:** Comment describing code intent.

**261 |** cm = confusion\_matrix(true\_labels, predicted\_labels)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**263 |** plt.figure(figsize=(10, 8))

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**265 |** sns.heatmap(

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**267 |**  cm,

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**269 |**  annot=True,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**271 |**  fmt='d',

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**273 |**  cmap='Blues',

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**275 |**  xticklabels=label\_encoder.classes\_,

**Explication:** Encode string labels to integer IDs.

**Raison / Choix:** Required by ML frameworks which expect numeric labels for classification.

**277 |**  yticklabels=label\_encoder.classes\_

**Explication:** Encode string labels to integer IDs.

**Raison / Choix:** Required by ML frameworks which expect numeric labels for classification.

**279 |** )

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**281 |** plt.title('Matrice de Confusion - Intent Classification')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**283 |** plt.ylabel('Vraie Classe')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**285 |** plt.xlabel('Classe Predite')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**287 |** plt.tight\_layout()

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**289 |** plt.show()

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**293 |** # Sauvegarder le modele

**Explication:** Comment: Sauvegarder le modele

**Raison / Choix:** Comment describing code intent.

**295 |** model\_path = os.path.join(workdir, 'models/intent\_classifier\_best')

**Explication:** Create / join filesystem paths.

**Raison / Choix:** Set up working directories consistently (works on Colab and local).

**297 |** intent\_trainer.save\_model(model\_path)

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**299 |** tokenizer.save\_pretrained(model\_path)

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**303 |** print(f'\nModele sauvegarde dans: {model\_path}')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

## Cellule de code 8

**Code:**

from transformers import AutoTokenizer  
  
  
  
# Tokenizer rapide pour NER  
  
tokenizer\_fast = AutoTokenizer.from\_pretrained('camembert-base', use\_fast=True)  
  
  
  
# Labels NER en format BIO  
  
ner\_labels = ['O', 'B-Departure', 'I-Departure', 'B-Destination', 'I-Destination']  
  
label2id = {label: i for i, label in enumerate(ner\_labels)}  
  
id2label = {i: label for label, i in label2id.items()}  
  
  
  
print('Labels NER:')  
  
for label\_id, label\_name in id2label.items():  
  
 print(f' {label\_id}: {label\_name}')  
  
  
  
def convert\_to\_bio\_tags(text, entities):  
  
 """Convertit les annotations en tags BIO pour chaque token."""  
  
 encoding = tokenizer\_fast(  
  
 text,  
  
 return\_offsets\_mapping=True,  
  
 truncation=True,  
  
 max\_length=128,  
  
 padding='max\_length'  
  
 )  
  
  
  
 offset\_mapping = encoding['offset\_mapping']  
  
 labels = ['O'] \* len(offset\_mapping)  
  
  
  
 for entity in entities:  
  
 start\_char = entity['start']  
  
 end\_char = entity['end']  
  
 entity\_label = entity['label']  
  
  
  
 token\_indices = []  
  
 for idx, (token\_start, token\_end) in enumerate(offset\_mapping):  
  
 if token\_start == token\_end == 0:  
  
 continue  
  
 if not (token\_end <= start\_char or token\_start >= end\_char):  
  
 token\_indices.append(idx)  
  
  
  
 if token\_indices:  
  
 labels[token\_indices[0]] = f'B-{entity\_label}'  
  
 for idx in token\_indices[1:]:  
  
 labels[idx] = f'I-{entity\_label}'  
  
  
  
 label\_ids = [label2id.get(label, 0) for label in labels]  
  
  
  
 return {  
  
 'input\_ids': encoding['input\_ids'],  
  
 'attention\_mask': encoding['attention\_mask'],  
  
 'labels': label\_ids  
  
 }  
  
  
  
# Preparer les datasets NER  
  
print('\nConversion en format BIO...')  
  
ner\_train\_data = []  
  
for \_, row in train\_df.iterrows():  
  
 entities = row['parsed\_entities'] if row['intent'] == 'TRIP' else []  
  
 ner\_example = convert\_to\_bio\_tags(row['text'], entities)  
  
 ner\_train\_data.append(ner\_example)  
  
  
  
ner\_test\_data = []  
  
for \_, row in test\_df.iterrows():  
  
 entities = row['parsed\_entities'] if row['intent'] == 'TRIP' else []  
  
 ner\_example = convert\_to\_bio\_tags(row['text'], entities)  
  
 ner\_test\_data.append(ner\_example)  
  
  
  
ner\_train\_dataset = Dataset.from\_list(ner\_train\_data)  
  
ner\_test\_dataset = Dataset.from\_list(ner\_test\_data)  
  
  
  
print(f'Datasets NER crees:')  
  
print(f' Train: {len(ner\_train\_dataset)} exemples')  
  
print(f' Test: {len(ner\_test\_dataset)} exemples')

Explications ligne par ligne :

**001 |** from transformers import AutoTokenizer

**Explication:** Import Transformers components.

**Raison / Choix:** We use Hugging Face Transformers for tokenization, modelling and training.

**005 |** # Tokenizer rapide pour NER

**Explication:** Comment: Tokenizer rapide pour NER

**Raison / Choix:** Comment describing code intent.

**007 |** tokenizer\_fast = AutoTokenizer.from\_pretrained('camembert-base', use\_fast=True)

**Explication:** Load a pretrained tokenizer (CamemBERT).

**Raison / Choix:** Tokenization converts raw text to token ids compatible with the model; CamemBERT is a French pretrained model.

**011 |** # Labels NER en format BIO

**Explication:** Comment: Labels NER en format BIO

**Raison / Choix:** Comment describing code intent.

**013 |** ner\_labels = ['O', 'B-Departure', 'I-Departure', 'B-Destination', 'I-Destination']

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**015 |** label2id = {label: i for i, label in enumerate(ner\_labels)}

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**017 |** id2label = {i: label for label, i in label2id.items()}

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**021 |** print('Labels NER:')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**023 |** for label\_id, label\_name in id2label.items():

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**025 |**  print(f' {label\_id}: {label\_name}')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**029 |** def convert\_to\_bio\_tags(text, entities):

**Explication:** Define function.

**Raison / Choix:** Creates a reusable function used later in the notebook.

**031 |**  """Convertit les annotations en tags BIO pour chaque token."""

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**033 |**  encoding = tokenizer\_fast(

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**035 |**  text,

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**037 |**  return\_offsets\_mapping=True,

**Explication:** Use offset mapping during tokenization.

**Raison / Choix:** Necessary to align token indices with character offsets for BIO labeling.

**039 |**  truncation=True,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**041 |**  max\_length=128,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**043 |**  padding='max\_length'

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**045 |**  )

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**049 |**  offset\_mapping = encoding['offset\_mapping']

**Explication:** Use offset mapping during tokenization.

**Raison / Choix:** Necessary to align token indices with character offsets for BIO labeling.

**051 |**  labels = ['O'] \* len(offset\_mapping)

**Explication:** Use offset mapping during tokenization.

**Raison / Choix:** Necessary to align token indices with character offsets for BIO labeling.

**055 |**  for entity in entities:

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**057 |**  start\_char = entity['start']

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**059 |**  end\_char = entity['end']

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**061 |**  entity\_label = entity['label']

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**065 |**  token\_indices = []

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**067 |**  for idx, (token\_start, token\_end) in enumerate(offset\_mapping):

**Explication:** Use offset mapping during tokenization.

**Raison / Choix:** Necessary to align token indices with character offsets for BIO labeling.

**069 |**  if token\_start == token\_end == 0:

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**071 |**  continue

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**073 |**  if not (token\_end <= start\_char or token\_start >= end\_char):

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**075 |**  token\_indices.append(idx)

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**079 |**  if token\_indices:

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**081 |**  labels[token\_indices[0]] = f'B-{entity\_label}'

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**083 |**  for idx in token\_indices[1:]:

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**085 |**  labels[idx] = f'I-{entity\_label}'

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**089 |**  label\_ids = [label2id.get(label, 0) for label in labels]

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**093 |**  return {

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**095 |**  'input\_ids': encoding['input\_ids'],

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**097 |**  'attention\_mask': encoding['attention\_mask'],

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**099 |**  'labels': label\_ids

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**101 |**  }

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**105 |** # Preparer les datasets NER

**Explication:** Comment: Preparer les datasets NER

**Raison / Choix:** Comment describing code intent.

**107 |** print('\nConversion en format BIO...')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**109 |** ner\_train\_data = []

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**111 |** for \_, row in train\_df.iterrows():

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**113 |**  entities = row['parsed\_entities'] if row['intent'] == 'TRIP' else []

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**115 |**  ner\_example = convert\_to\_bio\_tags(row['text'], entities)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**117 |**  ner\_train\_data.append(ner\_example)

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**121 |** ner\_test\_data = []

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**123 |** for \_, row in test\_df.iterrows():

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**125 |**  entities = row['parsed\_entities'] if row['intent'] == 'TRIP' else []

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**127 |**  ner\_example = convert\_to\_bio\_tags(row['text'], entities)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**129 |**  ner\_test\_data.append(ner\_example)

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**133 |** ner\_train\_dataset = Dataset.from\_list(ner\_train\_data)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**135 |** ner\_test\_dataset = Dataset.from\_list(ner\_test\_data)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**139 |** print(f'Datasets NER crees:')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**141 |** print(f' Train: {len(ner\_train\_dataset)} exemples')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**143 |** print(f' Test: {len(ner\_test\_dataset)} exemples')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

## Cellule de code 9

**Code:**

from transformers import (  
  
 AutoModelForTokenClassification,  
  
 DataCollatorForTokenClassification,  
  
 TrainingArguments,  
  
 Trainer  
  
)  
  
import evaluate  
  
  
  
# Charger le modele CamemBERT pour Token Classification  
  
ner\_model = AutoModelForTokenClassification.from\_pretrained(  
  
 'camembert-base',  
  
 num\_labels=len(ner\_labels),  
  
 id2label=id2label,  
  
 label2id=label2id  
  
)  
  
  
  
# Data collator  
  
data\_collator = DataCollatorForTokenClassification(tokenizer\_fast)  
  
  
  
# Metrique seqeval  
  
seqeval\_metric = evaluate.load('seqeval')  
  
  
  
def compute\_ner\_metrics(eval\_pred):  
  
 """Calcule les metriques NER avec seqeval."""  
  
 logits, labels = eval\_pred  
  
 predictions = np.argmax(logits, axis=-1)  
  
  
  
 true\_labels = []  
  
 pred\_labels = []  
  
  
  
 for pred\_seq, label\_seq in zip(predictions, labels):  
  
 true\_tags = []  
  
 pred\_tags = []  
  
  
  
 for pred\_id, label\_id in zip(pred\_seq, label\_seq):  
  
 if label\_id != -100:  
  
 true\_tags.append(id2label[label\_id])  
  
 pred\_tags.append(id2label[pred\_id])  
  
  
  
 true\_labels.append(true\_tags)  
  
 pred\_labels.append(pred\_tags)  
  
  
  
 results = seqeval\_metric.compute(predictions=pred\_labels, references=true\_labels)  
  
  
  
 return {  
  
 'precision': results['overall\_precision'],  
  
 'recall': results['overall\_recall'],  
  
 'f1': results['overall\_f1'],  
  
 'accuracy': results['overall\_accuracy']  
  
 }  
  
  
  
# Arguments d'entrainement  
  
ner\_training\_args = TrainingArguments(  
  
 output\_dir=os.path.join(workdir, 'models/ner\_model'),  
  
 num\_train\_epochs=4,  
  
 per\_device\_train\_batch\_size=8,  
  
 per\_device\_eval\_batch\_size=16,  
  
 learning\_rate=3e-5,  
  
 weight\_decay=0.01,  
  
 eval\_strategy='epoch',  
  
 save\_strategy='epoch',  
  
 load\_best\_model\_at\_end=True,  
  
 metric\_for\_best\_model='f1',  
  
 logging\_steps=100,  
  
 warmup\_steps=200,  
  
 fp16=torch.cuda.is\_available(),  
  
 push\_to\_hub=False,  
  
 report\_to='none',  
  
)  
  
  
  
# Creer le Trainer  
  
ner\_trainer = Trainer(  
  
 model=ner\_model,  
  
 args=ner\_training\_args,  
  
 train\_dataset=ner\_train\_dataset,  
  
 eval\_dataset=ner\_test\_dataset,  
  
 tokenizer=tokenizer\_fast,  
  
 data\_collator=data\_collator,  
  
 compute\_metrics=compute\_ner\_metrics  
  
)  
  
  
  
print('\n' + '='\*70)  
  
print('DEBUT DU FINE-TUNING NER')  
  
print('='\*70)  
  
  
  
# Entrainer le modele  
  
ner\_trainer.train()  
  
  
  
print('\nFine-tuning NER termine!')

Explications ligne par ligne :

**001 |** from transformers import (

**Explication:** Import Transformers components.

**Raison / Choix:** We use Hugging Face Transformers for tokenization, modelling and training.

**003 |**  AutoModelForTokenClassification,

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**005 |**  DataCollatorForTokenClassification,

**Explication:** Load token-classification model and data collator.

**Raison / Choix:** Used for NER to tag tokens with BIO labels (Departure/Destination).

**007 |**  TrainingArguments,

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**009 |**  Trainer

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**011 |** )

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**013 |** import evaluate

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**017 |** # Charger le modele CamemBERT pour Token Classification

**Explication:** Comment: Charger le modele CamemBERT pour Token Classification

**Raison / Choix:** Comment describing code intent.

**019 |** ner\_model = AutoModelForTokenClassification.from\_pretrained(

**Explication:** Load token-classification model and data collator.

**Raison / Choix:** Used for NER to tag tokens with BIO labels (Departure/Destination).

**021 |**  'camembert-base',

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**023 |**  num\_labels=len(ner\_labels),

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**025 |**  id2label=id2label,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**027 |**  label2id=label2id

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**029 |** )

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**033 |** # Data collator

**Explication:** Comment: Data collator

**Raison / Choix:** Comment describing code intent.

**035 |** data\_collator = DataCollatorForTokenClassification(tokenizer\_fast)

**Explication:** Load token-classification model and data collator.

**Raison / Choix:** Used for NER to tag tokens with BIO labels (Departure/Destination).

**039 |** # Metrique seqeval

**Explication:** Comment: Metrique seqeval

**Raison / Choix:** Comment describing code intent.

**041 |** seqeval\_metric = evaluate.load('seqeval')

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**045 |** def compute\_ner\_metrics(eval\_pred):

**Explication:** Define function.

**Raison / Choix:** Creates a reusable function used later in the notebook.

**047 |**  """Calcule les metriques NER avec seqeval."""

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**049 |**  logits, labels = eval\_pred

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**051 |**  predictions = np.argmax(logits, axis=-1)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**055 |**  true\_labels = []

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**057 |**  pred\_labels = []

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**061 |**  for pred\_seq, label\_seq in zip(predictions, labels):

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**063 |**  true\_tags = []

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**065 |**  pred\_tags = []

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**069 |**  for pred\_id, label\_id in zip(pred\_seq, label\_seq):

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**071 |**  if label\_id != -100:

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**073 |**  true\_tags.append(id2label[label\_id])

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**075 |**  pred\_tags.append(id2label[pred\_id])

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**079 |**  true\_labels.append(true\_tags)

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**081 |**  pred\_labels.append(pred\_tags)

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**085 |**  results = seqeval\_metric.compute(predictions=pred\_labels, references=true\_labels)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**089 |**  return {

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**091 |**  'precision': results['overall\_precision'],

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**093 |**  'recall': results['overall\_recall'],

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**095 |**  'f1': results['overall\_f1'],

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**097 |**  'accuracy': results['overall\_accuracy']

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**099 |**  }

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**103 |** # Arguments d'entrainement

**Explication:** Comment: Arguments d'entrainement

**Raison / Choix:** Comment describing code intent.

**105 |** ner\_training\_args = TrainingArguments(

**Explication:** Define training hyperparameters.

**Raison / Choix:** Controls epochs, batch size, learning rate and other aspects of training stability and performance.

**107 |**  output\_dir=os.path.join(workdir, 'models/ner\_model'),

**Explication:** Create / join filesystem paths.

**Raison / Choix:** Set up working directories consistently (works on Colab and local).

**109 |**  num\_train\_epochs=4,

**Explication:** Define training hyperparameters.

**Raison / Choix:** Controls epochs, batch size, learning rate and other aspects of training stability and performance.

**111 |**  per\_device\_train\_batch\_size=8,

**Explication:** Define training hyperparameters.

**Raison / Choix:** Controls epochs, batch size, learning rate and other aspects of training stability and performance.

**113 |**  per\_device\_eval\_batch\_size=16,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**115 |**  learning\_rate=3e-5,

**Explication:** Define training hyperparameters.

**Raison / Choix:** Controls epochs, batch size, learning rate and other aspects of training stability and performance.

**117 |**  weight\_decay=0.01,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**119 |**  eval\_strategy='epoch',

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**121 |**  save\_strategy='epoch',

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**123 |**  load\_best\_model\_at\_end=True,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**125 |**  metric\_for\_best\_model='f1',

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**127 |**  logging\_steps=100,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**129 |**  warmup\_steps=200,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**131 |**  fp16=torch.cuda.is\_available(),

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**133 |**  push\_to\_hub=False,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**135 |**  report\_to='none',

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**137 |** )

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**141 |** # Creer le Trainer

**Explication:** Comment: Creer le Trainer

**Raison / Choix:** Comment describing code intent.

**143 |** ner\_trainer = Trainer(

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**145 |**  model=ner\_model,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**147 |**  args=ner\_training\_args,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**149 |**  train\_dataset=ner\_train\_dataset,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**151 |**  eval\_dataset=ner\_test\_dataset,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**153 |**  tokenizer=tokenizer\_fast,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**155 |**  data\_collator=data\_collator,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**157 |**  compute\_metrics=compute\_ner\_metrics

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**159 |** )

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**163 |** print('\n' + '='\*70)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**165 |** print('DEBUT DU FINE-TUNING NER')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**167 |** print('='\*70)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**171 |** # Entrainer le modele

**Explication:** Comment: Entrainer le modele

**Raison / Choix:** Comment describing code intent.

**173 |** ner\_trainer.train()

**Explication:** Start model fine-tuning.

**Raison / Choix:** Runs gradient updates on training data to adapt the pretrained model to the target task.

**177 |** print('\nFine-tuning NER termine!')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

## Cellule de code 10

**Code:**

from seqeval.metrics import classification\_report as seqeval\_report  
  
  
  
# Evaluer sur le test set  
  
ner\_eval\_results = ner\_trainer.evaluate()  
  
  
  
print('='\*70)  
  
print('RESULTATS NER SUR LE TEST SET')  
  
print('='\*70)  
  
for key, value in ner\_eval\_results.items():  
  
 if isinstance(value, float):  
  
 print(f' {key}: {value:.4f}')  
  
  
  
# Predictions detaillees  
  
ner\_predictions = ner\_trainer.predict(ner\_test\_dataset)  
  
predicted\_logits = ner\_predictions.predictions  
  
predicted\_labels = np.argmax(predicted\_logits, axis=-1)  
  
true\_labels = ner\_predictions.label\_ids  
  
  
  
# Convertir en tags pour seqeval  
  
true\_tags\_list = []  
  
pred\_tags\_list = []  
  
  
  
for pred\_seq, label\_seq in zip(predicted\_labels, true\_labels):  
  
 true\_tags = []  
  
 pred\_tags = []  
  
  
  
 for pred\_id, label\_id in zip(pred\_seq, label\_seq):  
  
 if label\_id != -100:  
  
 true\_tags.append(id2label[label\_id])  
  
 pred\_tags.append(id2label[pred\_id])  
  
  
  
 true\_tags\_list.append(true\_tags)  
  
 pred\_tags\_list.append(pred\_tags)  
  
  
  
# Classification report detaille  
  
print('\n' + '='\*70)  
  
print('CLASSIFICATION REPORT NER (par entite)')  
  
print('='\*70)  
  
print(seqeval\_report(true\_tags\_list, pred\_tags\_list, digits=4))  
  
  
  
# Sauvegarder le modele  
  
ner\_model\_path = os.path.join(workdir, 'models/ner\_model\_best')  
  
ner\_trainer.save\_model(ner\_model\_path)  
  
tokenizer\_fast.save\_pretrained(ner\_model\_path)  
  
  
  
print(f'\nModele NER sauvegarde dans: {ner\_model\_path}')

Explications ligne par ligne :

**001 |** from seqeval.metrics import classification\_report as seqeval\_report

**Explication:** Evaluate the model on the test set and compute metrics.

**Raison / Choix:** Gives objective performance measures like accuracy and F1 to assess model quality.

**005 |** # Evaluer sur le test set

**Explication:** Comment: Evaluer sur le test set

**Raison / Choix:** Comment describing code intent.

**007 |** ner\_eval\_results = ner\_trainer.evaluate()

**Explication:** Evaluate the model on the test set and compute metrics.

**Raison / Choix:** Gives objective performance measures like accuracy and F1 to assess model quality.

**011 |** print('='\*70)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**013 |** print('RESULTATS NER SUR LE TEST SET')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**015 |** print('='\*70)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**017 |** for key, value in ner\_eval\_results.items():

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**019 |**  if isinstance(value, float):

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**021 |**  print(f' {key}: {value:.4f}')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**025 |** # Predictions detaillees

**Explication:** Comment: Predictions detaillees

**Raison / Choix:** Comment describing code intent.

**027 |** ner\_predictions = ner\_trainer.predict(ner\_test\_dataset)

**Explication:** Evaluate the model on the test set and compute metrics.

**Raison / Choix:** Gives objective performance measures like accuracy and F1 to assess model quality.

**029 |** predicted\_logits = ner\_predictions.predictions

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**031 |** predicted\_labels = np.argmax(predicted\_logits, axis=-1)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**033 |** true\_labels = ner\_predictions.label\_ids

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**037 |** # Convertir en tags pour seqeval

**Explication:** Comment: Convertir en tags pour seqeval

**Raison / Choix:** Comment describing code intent.

**039 |** true\_tags\_list = []

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**041 |** pred\_tags\_list = []

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**045 |** for pred\_seq, label\_seq in zip(predicted\_labels, true\_labels):

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**047 |**  true\_tags = []

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**049 |**  pred\_tags = []

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**053 |**  for pred\_id, label\_id in zip(pred\_seq, label\_seq):

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**055 |**  if label\_id != -100:

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**057 |**  true\_tags.append(id2label[label\_id])

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**059 |**  pred\_tags.append(id2label[pred\_id])

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**063 |**  true\_tags\_list.append(true\_tags)

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**065 |**  pred\_tags\_list.append(pred\_tags)

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**069 |** # Classification report detaille

**Explication:** Comment: Classification report detaille

**Raison / Choix:** Comment describing code intent.

**071 |** print('\n' + '='\*70)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**073 |** print('CLASSIFICATION REPORT NER (par entite)')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**075 |** print('='\*70)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**077 |** print(seqeval\_report(true\_tags\_list, pred\_tags\_list, digits=4))

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**081 |** # Sauvegarder le modele

**Explication:** Comment: Sauvegarder le modele

**Raison / Choix:** Comment describing code intent.

**083 |** ner\_model\_path = os.path.join(workdir, 'models/ner\_model\_best')

**Explication:** Create / join filesystem paths.

**Raison / Choix:** Set up working directories consistently (works on Colab and local).

**085 |** ner\_trainer.save\_model(ner\_model\_path)

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**087 |** tokenizer\_fast.save\_pretrained(ner\_model\_path)

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**091 |** print(f'\nModele NER sauvegarde dans: {ner\_model\_path}')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

## Cellule de code 11

**Code:**

from transformers import pipeline  
  
  
  
# Charger les pipelines  
  
print('Chargement des modeles...')  
  
  
  
intent\_model\_path = os.path.join(workdir, 'models/intent\_classifier\_best')  
  
ner\_model\_path = os.path.join(workdir, 'models/ner\_model\_best')  
  
  
  
intent\_pipeline = pipeline(  
  
 'text-classification',  
  
 model=intent\_model\_path,  
  
 tokenizer=intent\_model\_path,  
  
 device=0 if torch.cuda.is\_available() else -1  
  
)  
  
  
  
ner\_pipeline = pipeline(  
  
 'token-classification',  
  
 model=ner\_model\_path,  
  
 tokenizer=ner\_model\_path,  
  
 aggregation\_strategy='simple',  
  
 device=0 if torch.cuda.is\_available() else -1  
  
)  
  
  
  
print('Pipelines charges!')  
  
  
  
def predict\_travel\_order(sentence\_id, text):  
  
 """Pipeline complet : Intent + NER"""  
  
 # 1. Predire l'intent  
  
 intent\_result = intent\_pipeline(text)[0]  
  
 predicted\_intent = intent\_result['label']  
  
 confidence = intent\_result['score']  
  
  
  
 # 2. Si pas TRIP, retourner l'intent  
  
 if predicted\_intent != 'TRIP':  
  
 return f"{sentence\_id},{predicted\_intent}", confidence  
  
  
  
 # 3. Si TRIP, extraire les entites  
  
 ner\_results = ner\_pipeline(text)  
  
  
  
 departure = None  
  
 destination = None  
  
  
  
 for entity in ner\_results:  
  
 entity\_label = entity.get('entity\_group', entity.get('entity'))  
  
 word = entity['word'].replace('▁', ' ').strip()  
  
  
  
 if 'Departure' in entity\_label and not departure:  
  
 departure = word  
  
 elif 'Destination' in entity\_label and not destination:  
  
 destination = word  
  
  
  
 if departure and destination:  
  
 return f"{sentence\_id},{departure},{destination}", confidence  
  
 else:  
  
 return f"{sentence\_id},UNKNOWN", confidence  
  
  
  
# Tester sur les exemples problematiques mentionnes  
  
print('\n' + '='\*70)  
  
print('TESTS SUR EXEMPLES PROBLEMATIQUES')  
  
print('='\*70)  
  
  
  
test\_examples = [  
  
 ("4740", "Bonjour, je compte faire le trajet vendredi de Cotonou à Porto-Novo"),  
  
 ("5606", "À quelle heure y a-t-il des trains de Martigues à Lyon ?"),  
  
 ("4824", "Comment aller de Malanville à Caen ?"),  
  
 ("4205", "Is there a train to Manchester?"),  
  
 ("3228", "de Semur-en-Auxois à Dreux"),  
  
 ("2745", "Billet Les Sables-d'Olonne à Château-Chinon")  
  
]  
  
  
  
for sid, text in test\_examples:  
  
 result, conf = predict\_travel\_order(sid, text)  
  
 print(f'\n{sid}: {text}')  
  
 print(f' -> Prediction: {result}')  
  
 print(f' -> Confiance: {conf:.2%}')

Explications ligne par ligne :

**001 |** from transformers import pipeline

**Explication:** Import Transformers components.

**Raison / Choix:** We use Hugging Face Transformers for tokenization, modelling and training.

**005 |** # Charger les pipelines

**Explication:** Comment: Charger les pipelines

**Raison / Choix:** Comment describing code intent.

**007 |** print('Chargement des modeles...')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**011 |** intent\_model\_path = os.path.join(workdir, 'models/intent\_classifier\_best')

**Explication:** Create / join filesystem paths.

**Raison / Choix:** Set up working directories consistently (works on Colab and local).

**013 |** ner\_model\_path = os.path.join(workdir, 'models/ner\_model\_best')

**Explication:** Create / join filesystem paths.

**Raison / Choix:** Set up working directories consistently (works on Colab and local).

**017 |** intent\_pipeline = pipeline(

**Explication:** Create Hugging Face inference pipelines.

**Raison / Choix:** Simplifies running the trained models for intent classification and NER during inference.

**019 |**  'text-classification',

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**021 |**  model=intent\_model\_path,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**023 |**  tokenizer=intent\_model\_path,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**025 |**  device=0 if torch.cuda.is\_available() else -1

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**027 |** )

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**031 |** ner\_pipeline = pipeline(

**Explication:** Create Hugging Face inference pipelines.

**Raison / Choix:** Simplifies running the trained models for intent classification and NER during inference.

**033 |**  'token-classification',

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**035 |**  model=ner\_model\_path,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**037 |**  tokenizer=ner\_model\_path,

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**039 |**  aggregation\_strategy='simple',

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**041 |**  device=0 if torch.cuda.is\_available() else -1

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**043 |** )

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**047 |** print('Pipelines charges!')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**051 |** def predict\_travel\_order(sentence\_id, text):

**Explication:** Define function.

**Raison / Choix:** Creates a reusable function used later in the notebook.

**053 |**  """Pipeline complet : Intent + NER"""

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**055 |**  # 1. Predire l'intent

**Explication:** Comment: 1. Predire l'intent

**Raison / Choix:** Comment describing code intent.

**057 |**  intent\_result = intent\_pipeline(text)[0]

**Explication:** Create Hugging Face inference pipelines.

**Raison / Choix:** Simplifies running the trained models for intent classification and NER during inference.

**059 |**  predicted\_intent = intent\_result['label']

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**061 |**  confidence = intent\_result['score']

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**065 |**  # 2. Si pas TRIP, retourner l'intent

**Explication:** Comment: 2. Si pas TRIP, retourner l'intent

**Raison / Choix:** Comment describing code intent.

**067 |**  if predicted\_intent != 'TRIP':

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**069 |**  return f"{sentence\_id},{predicted\_intent}", confidence

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**073 |**  # 3. Si TRIP, extraire les entites

**Explication:** Comment: 3. Si TRIP, extraire les entites

**Raison / Choix:** Comment describing code intent.

**075 |**  ner\_results = ner\_pipeline(text)

**Explication:** Create Hugging Face inference pipelines.

**Raison / Choix:** Simplifies running the trained models for intent classification and NER during inference.

**079 |**  departure = None

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**081 |**  destination = None

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**085 |**  for entity in ner\_results:

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**087 |**  entity\_label = entity.get('entity\_group', entity.get('entity'))

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**089 |**  word = entity['word'].replace('▁', ' ').strip()

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**093 |**  if 'Departure' in entity\_label and not departure:

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**095 |**  departure = word

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**097 |**  elif 'Destination' in entity\_label and not destination:

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**099 |**  destination = word

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**103 |**  if departure and destination:

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**105 |**  return f"{sentence\_id},{departure},{destination}", confidence

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**107 |**  else:

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**109 |**  return f"{sentence\_id},UNKNOWN", confidence

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**113 |** # Tester sur les exemples problematiques mentionnes

**Explication:** Comment: Tester sur les exemples problematiques mentionnes

**Raison / Choix:** Comment describing code intent.

**115 |** print('\n' + '='\*70)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**117 |** print('TESTS SUR EXEMPLES PROBLEMATIQUES')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**119 |** print('='\*70)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**123 |** test\_examples = [

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**125 |**  ("4740", "Bonjour, je compte faire le trajet vendredi de Cotonou à Porto-Novo"),

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**127 |**  ("5606", "À quelle heure y a-t-il des trains de Martigues à Lyon ?"),

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**129 |**  ("4824", "Comment aller de Malanville à Caen ?"),

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**131 |**  ("4205", "Is there a train to Manchester?"),

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**133 |**  ("3228", "de Semur-en-Auxois à Dreux"),

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**135 |**  ("2745", "Billet Les Sables-d'Olonne à Château-Chinon")

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**137 |** ]

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**141 |** for sid, text in test\_examples:

**Explication:** Control flow statement.

**Raison / Choix:** Alters execution path (loop/conditional).

**143 |**  result, conf = predict\_travel\_order(sid, text)

**Explication:** Assign a value to a variable.

**Raison / Choix:** Stores computation results or config values for later use.

**145 |**  print(f'\n{sid}: {text}')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**147 |**  print(f' -> Prediction: {result}')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.

**149 |**  print(f' -> Confiance: {conf:.2%}')

**Explication:** Execute a Python statement.

**Raison / Choix:** Line performs a standard Python operation; explanation omitted because it was simple or repetitive.